Theoretical Computer Science 411 (2010) 3923–3931

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A metric for rooted trees with unlabeled vertices based on nested parentheses☆

Chan-Shuo Wu, Guan-Shieng Huang*

Department of Computer Science and Information Engineering, National Chi Nan University, Taiwan

ARTICLE INFO

Article history:
Received 17 August 2009
Received in revised form 29 May 2010
Accepted 3 August 2010
Communicated by H. Prodinger

Keywords:
Tree metric
Nested parenthesis strings
NP-completeness
Approximation algorithm

ABSTRACT

In this paper, we propose a new metric for rooted trees with unlabeled vertices based on alignments of nested parenthesis strings. We prove that the time complexity for computing this metric is NP-hard and present a 1.5-approximation algorithm for it.

1. Introduction

Nested parentheses are widely studied and used in computer science, partly for the reason that they provide a convenient way to represent forests and trees by strings. For example, the Newick tree format, which was designed to represent phylogenetic trees in computational biology, is based on this idea [11]. More information about applications and properties of nested parentheses can be found in [19].

In this paper, we propose a metric for measuring the shape difference between two rooted unlabeled trees with the same number of nodes. The input trees are translated into nested parenthesis strings, on which the comparison is performed. Informally speaking, this metric is defined as the minimum number of *parenthesis moves* that can transform one nested parenthesis string into another. Its precise definition will be given in Section 2. We focus on the complexity issue for computing this metric. Although its definition is simple and natural, we will show that the time complexity is actually NP-hard. Hence we propose a 1.5-approximation algorithm for it in this paper.

There are several approaches that can measure the dissimilarity (or proximity) of two trees. Here, we discuss two of them related to this paper. The first approach is called the shortest path approach. It defines a set of tree transformation rules, such as left rotations and right rotations (see Fig. 1). Put the trees being considered as vertices of a graph, where two vertices are linked by an arc if and only if one vertex, which is a tree, can be transformed into another by applying any of the transformation rules once. This metric measures the length of the shortest path between any two vertices in this graph. This approach yields several important applications in tree comparisons, such as providing the tree edit distance [31] for arbitrary rooted trees and the tree rotation distance [30] for rooted binary trees with unlabeled vertices. For surveys and more recent results concerning tree edit distance, see [1,4,24,32]. Here we emphasize more the tree rotation distance. There is no known polynomial-time algorithm for evaluating the exact values of rotation distances, and even its NP-hardness has been open for more than two decades [6,30]. Therefore, some researchers turned to discussing restricted versions [5,29]

☆ This work was supported in part by the National Science Council of the Republic of China, Taiwan (grant no. NSC 98-2221-E-260-014).
* Corresponding author. Tel.: +886 49 2910960.
E-mail addresses: chanshuo@gate.sinica.edu.tw (C.-S. Wu), shieng@ncnu.edu.tw (G.-S. Huang).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.08.003

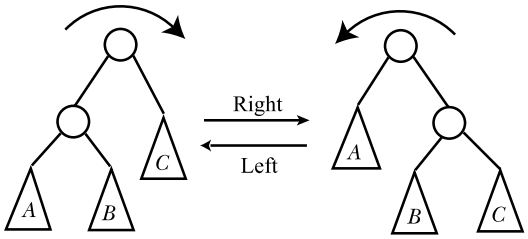


Fig. 1. Tree rotations.

or variants [7,15,27,34] of rotation distance, and others focused on estimating (numerical or algorithmic) upper and lower bounds [23,25,26,28,30] for this metric. More references on tree rotation distance can be found in [3,8–10,16,20,22,28].

The second approach is based on the idea of aligning two trees so that their corresponding similar parts can be identified and associated. An alignment is constructed under some constraints. Usually, it must be consistent with the structures of the two compared trees. Even so, there are too many ways to align them, and thus, a cost function that can judge the quality of an alignment is defined so that a best alignment, which has the minimum cost, can be chosen. A substitution matrix is used to weigh the cost for aligning two nodes of the trees, and the cost for the whole alignment is evaluated as the sum of costs for aligning all nodes in this alignment. The tree alignment distance [18] is an instance.

Our method in this paper follows the alignment approach. Let $\mathcal{T}_n$ be the set of all rooted trees with n unlabeled vertices. For any $T \in \mathcal{T}_n$, its parenthesis representation $\pi(T)$ can be obtained recursively as follows:

1. if T is a tree with only one node, $\pi(T) = ()$;
2. if the root of T is r_T and $T_1, \ldots, T_k$ are children of r_T in this order, then $\pi(T) = (\pi(T_1) \ldots \pi(T_k))$ for $k \geq 1$.

To compare the distance between any two trees $T_1, T_2 \in \mathcal{T}_n$, we compare the distance between $\pi(T_1)$ and $\pi(T_2)$. An optimal alignment for $\pi(T_1)$ and $\pi(T_2)$ is constructed in order to get the distance.

Inspired by the work of Jiang et al. [17], we originally proposed this metric while studying how to compare RNA secondary structures. However, after successive failures in designing a polynomial-time algorithm using dynamic programming, we realized that our current formulation could be NP-hard. Then following the research lines of [21], we successfully proved the NP-hardness and learned that it is necessary to circumvent this pitfall in order to develop a practical model for RNA secondary structure comparisons. This goal was achieved and the result is described in [33].

The organization of this paper is as follows. Section 2 formally defines the parenthesis rearrangement distance problem, and Section 3 proves its NP-hardness. Afterward, Section 4 provides a 1.5-approximation algorithm, and finally, Section 5 concludes this study.

2. The parenthesis rearrangement distance problem

A *parenthesis string* is a finite string over the symbols '(' and ')'. A parenthesis string $a_1 \ldots a_{2n}$ is *properly nested* when it contains n occurrences of '(' and n occurrences of ')', where the kth '(' precedes the kth ')' for $1 \leq k \leq n$. Let $\mathcal{P}_n$ be the set of properly nested parenthesis strings of length $2n$. For $P \in \mathcal{P}_n$, we use $P[i]$ to refer to the ith parenthesis of P and $P[i \ldots j]$ to refer to the substring of P from position i to position j. We say that $P[i]$ *pairs with* $P[j]$ where $1 \leq i < j \leq 2n$ iff (1) $P[i]$ is a left parenthesis and $P[j]$ is a right parenthesis; and (2) $P[i+1 \ldots j-1]$ is properly nested (including the empty string). In this case, we say that $P[i]$ and $P[j]$ form a pair, and call $P[i]$ a *mate* of $P[j]$ and vice versa.

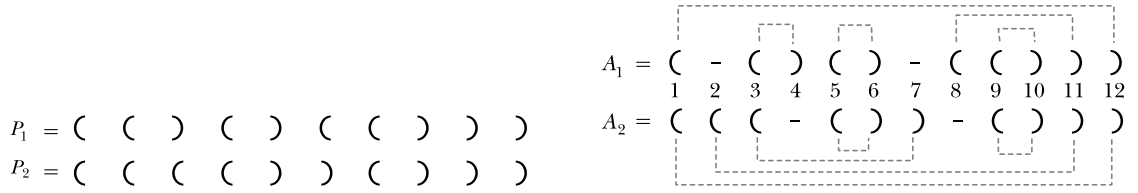
Before introducing our main problem, we define the concept of *parenthesis alignments* for nested parenthesis strings. For $P_1, P_2 \in \mathcal{P}_n$, we say that (A_1, A_2) is an alignment for P_1 and P_2 where $A_1 = a_1[1] \ldots a_1[m]$ and $A_2 = a_2[1] \ldots a_2[m]$ if and only if the following conditions hold:

PA1 The strings A_1 and A_2 are obtained from P_1 and P_2, respectively, by inserting spaces. This implies $m \geq 2n$.
PA2 If $a_1[i]$ and $a_2[i]$ are parentheses, they are of the same type (i.e. both are left parentheses or both are right parentheses).
PA3 Suppose $a_1[i]$, $a_1[j]$, $a_2[i]$ and $a_2[j]$ are parentheses where $1 \leq i < j \leq m$. Then $a_1[i]$ and $a_1[j]$ form a pair if and only if $a_2[i]$ and $a_2[j]$ form a pair.

In the above definition, we say that $a_1[i]$ is *aligned with* $a_2[i]$ for $1 \leq i \leq m$. If $a_1[i]$ and $a_2[i]$ are parentheses of the same type, then it is said that there is a *match* occurring at position i in this alignment; otherwise there is a *mismatch*.

Example 1. Let us consider the instance in Fig. 2. We use a dotted line to link two parentheses that form a pair. A space is symbolized by a minus sign. Notice that a match occurs at position 1 and a mismatch occurs at position 2 in the alignment (A_1, A_2). There are in total eight matches (at positions 1, 3, 5, 6, 9, 10, 11, 12) and four mismatches (at positions 2, 4, 7, 8).

The rearrangement distance for an alignment (A_1, A_2), denoted by $D(A_1, A_2)$, is defined as the number of mismatched parentheses in A_1, which is equal to the number of mismatched parentheses in A_2. The rearrangement distance $D(P_1, P_2)$ for two nested parenthesis strings P_1 and P_2 is defined as the minimum value of $D(A_1, A_2)$ over all possible alignments (A_1, A_2)



(a) Two parenthesis strings. (b) The alignment of P_1 and P_2 in (a).

Fig. 2. An example of the parenthesis alignment.

for P_1 and P_2 according to PA1–PA3. Intuitively, the function $D(P_1, P_2)$ accounts for the least number of parentheses that have to be moved in P_1 in order for it to become the same as P_2, under the constraint that pairs of parentheses in P_1 are still pairs in P_2. We use the shorthand PRD (which stands for the Parenthesis Rearrangement Distance) to refer to the problem of evaluating $D(P_1, P_2)$.

Example 2. Let us revisit Example 1. The alignment (A_1, A_2) in Fig. 2(b) is an optimal alignment for P_1 and P_2 in Fig. 2(a). From this alignment, we can see that $A_1[4]$ and $A_1[8]$ are aligned with spaces. Therefore, we can simultaneously move $A_1[4]$ to $A_2[7]$ and $A_1[8]$ to $A_2[2]$ so that P_1 becomes P_2. Hence, $D(P_1, P_2) = 2$.

Surprisingly, the PRD problem is NP-hard. We will give an NP-hardness proof for it in Section 3. In the rest of this section, we show that the parenthesis rearrangement distance fulfills the requirements for being a metric.

Theorem 1. *Let P_1, P_2 and P_3 be any nested parenthesis strings whose lengths are all $2n$. Then we have the following properties for P_1, P_2 and P_3:*

1. *The positive definiteness: $D(P_1, P_2) \geq 0$. In addition, $D(P_1, P_2) = 0$ if and only if P_1 and P_2 are identical.*
2. *The symmetry: $D(P_1, P_2) = D(P_2, P_1)$.*
3. *The triangle inequality: $D(P_1, P_2) + D(P_2, P_3) \geq D(P_1, P_3)$.*

Proof. Parts (1) and (2) come directly from the definition. As for Part (3), we will show that there exists an alignment (A_1, A_3) for P_1 and P_3 such that $D(P_1, P_2) + D(P_2, P_3) \geq D(A_1, A_3)$. Combining this with the fact that $D(A_1, A_3) \geq D(P_1, P_3)$, (3) will be established. The construction of (A_1, A_3) is as follows. Let (B_1, B_2) and (C_2, C_3) be alignments for P_1, P_2 and P_2, P_3 respectively such that $D(B_1, B_2) = D(P_1, P_2)$ and $D(C_2, C_3) = D(P_2, P_3)$. Then set $P_1[i]$ (the ith parenthesis in P_1) to be aligned with $P_3[k]$ in (A_1, A_3) if and only if $P_1[i]$ is aligned with $P_2[j]$ in (B_1, B_2) and $P_2[j]$ is aligned with $P_3[k]$ in (C_2, C_3) for some j where $1 \leq i, j, k \leq 2n$. Or equivalently, $P_1[i]$ is aligned with a space in (A_1, A_3) (and thus contributes 1 to $D(A_1, A_3)$) if and only if one of the following cases holds:

Case 1 : The symbol $P_1[i]$ is aligned with a space in (B_1, B_2). This contributes 1 to $D(P_1, P_2)$.
Case 2 : The symbol $P_1[i]$ is aligned with $P_2[j]$ in (B_1, B_2) for some j, but $P_2[j]$ is aligned with a space in (C_2, C_3). In this case, the latter condition contributes 1 to $D(P_2, P_3)$.

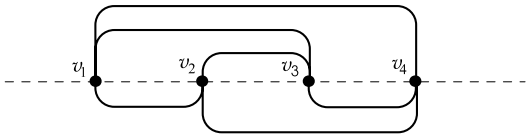
When some $P_1[i]$ contributes 1 to $D(A_1, A_3)$, it must be aligned with a space. Hence, it contributes 1 either to $D(P_1, P_2)$ or to $D(P_2, P_3)$. Therefore, we have $D(P_1, P_2) + D(P_2, P_3) \geq D(A_1, A_3)$. The inequality strictly holds when there exists $P_2[j]$ that is aligned with a space in (B_1, B_2) and a space in (C_2, C_3). In this case, $P_2[j]$ contributes 1 to $D(P_2, P_3)$ but there is no corresponding $P_1[i]$ that can share this cost in $D(A_1, A_3)$. □

3. An NP-hardness proof

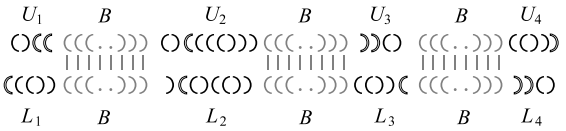
Let the DPRD problem be the decision version of the PRD problem. That is, we are given a parameter K in addition to two nested parenthesis strings P_1 and P_2, and want to determine whether $D(P_1, P_2) \leq K$. It is easy to see that the DPRD problem is in NP. In this section, we will show that the DPRD problem is NP-complete.

Our strategy for proving the NP-completeness is as follows. We consider its dual problem, and then show that this dual problem is NP-complete. Let (A_1, A_2) be an alignment for $P_1, P_2 \in \mathcal{P}_n$. Define $S(A_1, A_2)$ to be the number of matched parentheses for A_1 and A_2. It follows that $D(A_1, A_2) + S(A_1, A_2) = 2n$. Similarly, this definition can be extended to parenthesis strings, by considering the optimal alignment that can maximize $S(A_1, A_2)$. The same notation $S(P_1, P_2)$ is used to refer to this optimal value. Hence the identity $D(P_1, P_2) + S(P_1, P_2) = 2n$ holds. The evaluation of $S(P_1, P_2)$ is referred to as *the parenthesis rearrangement score problem*, or the PRS problem for short, throughout this paper. Let DPRS be the decision version of PRS. It follows that $D(P_1, P_2) \leq K$ if and only if $S(P_1, P_2) \geq 2n - K$. Hence, the complexities for these two problems are the same.

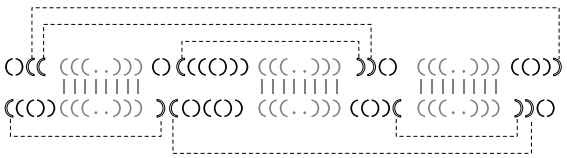
Before showing a key lemma from [21], we introduce some preliminary knowledge on two-page graphs. A graph is called a two-page graph if its vertices can be put on a straight line in an orderly fashion such that no edges are crossing. A two-page graph is called *cubic* if and only if every vertex of it has degree 3, and furthermore, there are at least one upward edge and one downward edge (see Fig. 3(a)).



(a) A 3-regular two-page graph G.



(b) The output of the reduction from (a).



(c) The 'edges' of G on (b).

Fig. 3. An illustration of the reduction.

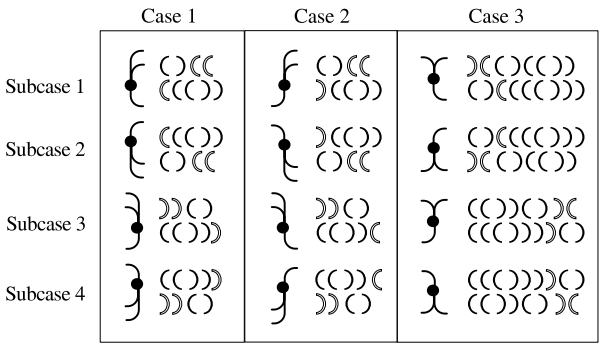


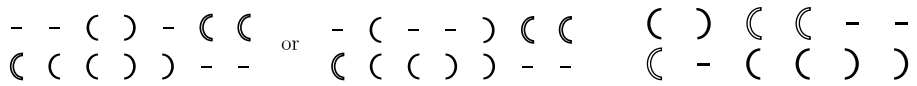
Fig. 4. The gadgets for the reduction to the DPRS problem.

Lemma 1 ([21]). *The maximum independent set problem restricted to simple cubic 2-page graphs is NP-hard.*

Lemma 1 is built on the results from [12–14,35]. In the following paragraphs, we are going to show that the DPRS problem is NP-complete.

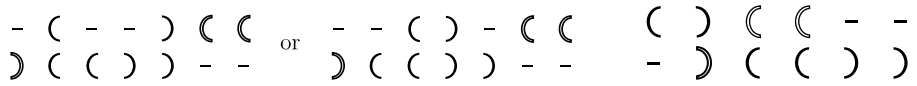
We will show how to reduce the maximum independent set problem from simple cubic two-page graphs to the DPRS problem. Let G be such a graph with n vertices; see Fig. 3(a) for an example. We classify vertices of G into three cases, and each case is further divided into four subcases, as shown in Fig. 4. The reduction starts from a linear scan on vertices of G from left to right, and finally two nested parenthesis strings S_1 and S_2 will be output.

Suppose the vertices of G are labeled in an orderly fashion by v_i for $1 \leq i \leq n$ from left to right. For each v_i, we look at Fig. 4 and get the corresponding upper block U_i and lower block L_i. Set $S'_1 = \prod_{1\leq i\leq n} BU_i$ and $S'_2 = \prod_{1\leq i\leq n} BL_i$ where B consists of $10n$ left parentheses followed by $10n$ right parentheses. The purpose of these B blocks is to serve as delimiters that can force an optimal alignment to align U_i with L_i for $1 \leq i \leq n$ (see Fig. 3(b) for an illustration and Lemma 3 for a formal proof). Notice that the lengths of S'_1 and S'_2 may not be equal. In fact, if v_i has one upward edge and two downward edges, $|U_i| - |L_i| = 1$; conversely, if v_i has two upward edges and one downward edge, $|U_i| - |L_i| = -1$. Therefore, we have to add more parentheses in order to make the lengths of S'_1 and S'_2 equal. Let $\delta = S'_1 - S'_2$. Since every vertex of G has odd degree, the total number of vertices of G must be even, which implies that δ is even. When $\delta > 0$, let H_1 be the empty string and H_2 consist of $\frac{\delta}{2}$ left parentheses followed by $\frac{\delta}{2}$ right parentheses. Otherwise, for $\delta \leq 0$, let H_1 consist of $\frac{-\delta}{2}$ left parentheses followed by $\frac{-\delta}{2}$ right parentheses and H_2 be the empty string. Let S_1 be $H_1S'_1$ and S_2 be $H_2S'_2$. The output of the reduction is S_1 and S_2.



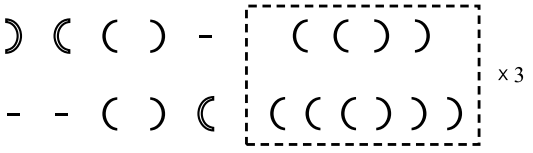
(a) The forbidden case. (b) The allowed case.

Fig. 5. Optimal alignments for Subcase 1.1 in Fig. 4.

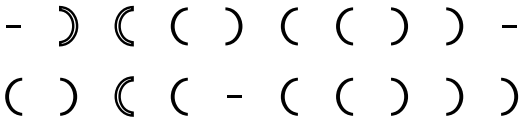


(a) The forbidden case. (b) The allowed case.

Fig. 6. Optimal alignments for Subcase 2.1 in Fig. 4.



(a) The forbidden case. The dotted-line box indicates that there are three ways to align the parentheses inside it; each way contributes four matches.



(b) The allowed case.

Fig. 7. Optimal alignments for Subcase 3.1 in Fig. 4.

Informally speaking, the complex U_i and L_i is a 'switch', which can mimic the effect of choosing or not choosing v_i for an independent set of G. This completes the specification of the reduction. For example, in Fig. 3(b), v_1–v_4 belong to Subcases 1.1, 3.2, 2.3, and 1.4, respectively. The output of the reduction for Fig. 3(a) is shown in Fig. 3(b).

We use the term *edge parenthesis* to refer to a parenthesis whose mate resides on a different block, such as those linked by dotted lines in Fig. 3(c); otherwise it is called a *nonedge parenthesis*. An edge parenthesis can be considered as one of the endpoints of an edge of G, and a pair of edge parentheses corresponds to a complete edge of G. Note that edge parentheses and nonedge parentheses are drawn as hollow shapes and solid shapes respectively in Figs. 3 and 4. We have the following lemma.

Lemma 2. *The maximum number of matches (constrained by PA1–PA3) of a U block and an L block in Fig. 4 depends on the following preconditions, according to whether an edge parenthesis inside U or L is allowed to be matched up or not:*

1. *Allowed: there are* 3 *matches for Cases* 1 *and* 2*, and* 7 *matches for Case* 3*.*
2. *Disallowed: there are* 2 *matches for Cases* 1 *and* 2*, and* 6 *matches for Case* 3*.*

Furthermore, in the allowed case, all *edge parentheses have to be matched up.*

Proof. We can exhaustively enumerate all maximum matchings of U and L for each subcase in Fig. 4. Note that Subcase 1.2 is a vertical reflection of Subcase 1.1, and Subcases 1.3 and 1.4 are horizontal reflections of Subcases 1.1 and 1.2 respectively. Hence, due to symmetry, we only need to consider Subcase 1.1 for Case 1. Similarly, we only need to consider Subcases 2.1 and 3.1 for Cases 2 and 3.

Let us first focus on Subcase 1.1. Suppose the edge parentheses cannot be matched up. In this case, there are exactly two nonedge parentheses in the U block. When all of them are matched up, we get two matches which is optimal, as shown in Fig. 5(a). Now consider the situation where the edge parentheses can be matched up. Notice that there are exactly four parentheses in the U block. However, block $U = \underline{()((}$ is not a subsequence of block $L = \underline{((()}$, and therefore not all parentheses in U can be matched up. Accordingly, there are at most three matches that can happen, and the result is shown in Fig. 5(b). In this case, all edge parentheses are matched up. This finishes the proof for Subcase 1.1.

The enumeration results for Subcases 2.1 and 3.1 are shown in Figs. 6 and 7, respectively. In these figures, panel (a)s consider the disallowed case and panel (b)s handle the allowed case. The dotted-line box in Fig. 7(a) indicates that there are three ways to align the parentheses inside it; each way contributes four matches. Furthermore, in the allowed case, all edge parenthesis are involved in the matchings. Therefore by arguments similar to those for Subcase 1.1, the claim follows. □

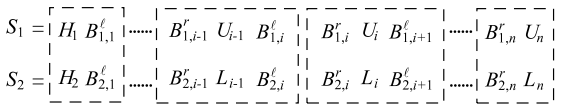


Fig. 8. The structure of the output of the reduction. Notice that the traces of any optimal alignment for S_1 and S_2 are limited to being within the dotted-line boxes.

Lemma 2 reveals a key point. Each complex U and L acts as a 'switch'. The 'switch on' indicates that all edge parentheses are matched up, and in this case, we acquire three matches for Cases 1 and 2, and get seven matches for Case 3 in Fig. 4. When the switch turns off, we always get one less match.

Recall that $S_1 = H_1 \prod_{1\le i\le n} BU_i$ and $S_2 = H_2 \prod_{1\le i\le n} BL_i$, which are the output of the reduction. In order to simplify the description of proof for Lemma 3, we set $U_0 = H_1$ and $L_0 = H_2$.

Lemma 3. *For any output of the reduction S_1 and S_2, there exists an optimal alignment that aligns U_i with L_i for $0 \le i \le n$.*

Proof. We use $B^{\ell}_{t,i}$ and $B^{r}_{t,i}$ to refer to the partition of the ith B block of S_t, for $t = 1, 2$, such that $B^{\ell}_{t,i}$ and $B^{r}_{t,i}$ contain only *left* parentheses and *right* parentheses, respectively. For any optimal alignment for S_1 and S_2, we first show that it cannot align a parenthesis of $B^{r}_{1,i}$ with a parenthesis of $B^{r}_{2,j}$ for $j \ne i$. If that was the case, let their positions be k_1 in S_1 and k_2 in S_2. In the following, we will show that the absolute value of $k_1 - k_2$ is very large, which implies that there are too many mismatches in this alignment so it cannot be optimal. Without loss of generality, assume $i > j$. Notice that the number of parentheses in each block listed in Fig. 4 is between 4 and 9. It follows that

$$\begin{aligned} k_1 &\ge \sum_{q=0}^{i-1} \left|U_q\right| + \sum_{q=1}^{i-1} \left|B^{\ell}_{1,q} B^{r}_{1,q}\right| + \left|B^{\ell}_{1,i}\right| \\ &\ge 4(i-1) + (i-1)\,|B| + \frac{|B|}{2}. \end{aligned} \tag{1}$$

Similarly, we have

$$k_2 \le \sum_{q=0}^{j-1} \left|L_q\right| + \sum_{q=1}^{j} \left|B^{\ell}_{1,q} B^{r}_{1,q}\right| \le n + 9(j-1) + j\,|B|\,. \tag{2}$$

When we subtract Eq. (2) from Eq. (1), we obtain

$$\begin{aligned} k_1 - k_2 &\ge 4(i-1) + (i-1)\,|B| + \frac{|B|}{2} - n - 9(j-1) - j\,|B| \\ &\ge \frac{|B|}{2} - n + |B|(i-1-j) + 4(i-1) - 9(j-1). \end{aligned} \tag{3}$$

Since $i > j$, the term $|B|(i-1-j) \ge 0$ and $4(i-1) \ge 4(j-1)$. Eq. (3) can be further simplified into

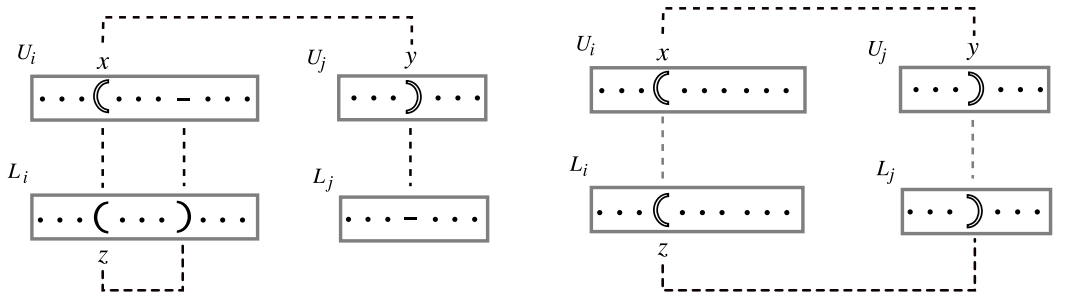
$$k_1 - k_2 \ge \frac{|B|}{2} - n - 5(j-1) \ge \frac{|B|}{2} - 6n = 10n - 6n = 4n. \tag{4}$$

Eq. (4) implies that there are at least $4n$ mismatched parentheses in S_1 for any alignment of S_1 and S_2 that aligns $S_1[k_1]$ with $S_2[k_2]$, since we have to insert at least $4n$ spaces into $S_2[1 \ldots k_2]$ in order to make its length equal to $S_1[1 \ldots k_1]$.

Consider the alignment A' that aligns U_i with L_i in the way shown in Figs. 5–7, and aligns $B^{\ell}_{1,i}$ with $B^{\ell}_{2,i}$ and $B^{r}_{1,i}$ with $B^{r}_{2,i}$ accordingly for $1 \le i \le n$. We would like to count an upper bound for the number of mismatched parentheses that occurred in S_1 for A'. Aligning H_1 with H_2 contributes at most n mismatched parentheses since the length of H_1 is at most n; aligning each U_i with L_i for $1 \le i \le n$ contributes at most 2, as shown in Figs. 5–7. To sum up, there are at most $n + 2n = 3n$ mismatched parentheses occurring in S_1 for A'. Therefore, the alignment that aligns $S_1[k_1]$ with $S_2[k_2]$ is worse than A', a contradiction.

The same argument can be extended to exclude the following two cases in an optimal alignment for S_1 and S_2: (1) aligning $B^{r}_{1,i}$ with L_j for $1 \le i \ne j \le n$; and (2) aligning $B^{r}_{1,1}$ with H_2. See Fig. 8 for an illustration. Combining these two cases with the original one, we conclude that $B^{r}_{1,i}$ can only be aligned with $B^{r}_{2,i}$ and L_i for $1 \le i \le n$ in an optimal alignment. A similar argument from symmetry would establish that $B^{\ell}_{1,i}$ can only be aligned with $B^{\ell}_{2,i}$ and L_{i-1} for $1 \le i \le n$ (recall that we set $L_0 = H_2$) in an optimal alignment.

As a summary, the above arguments reveal the fact that $B^{r}_{1,i} U_i B^{\ell}_{1,i+1}$ can only be aligned with $B^{r}_{2,i} L_i B^{\ell}_{2,i+1}$ for $1 \le i \le n-1$ in any optimal alignment for S_1 and S_2, as shown in Fig. 8. Since $B^{r}_{1,i}$ and $B^{\ell}_{1,i+1}$ are perfectly matched to $B^{r}_{2,i}$ and $B^{\ell}_{2,i+1}$, the minimum number of mismatches between $B^{r}_{1,i} U_i B^{\ell}_{1,i+1}$ and $B^{r}_{2,i} L_i B^{\ell}_{2,i+1}$ is equal to the minimum number of mismatches between U_i and L_i for $0 \le i \le n$. This establishes the claim of this lemma. □



(a) The symbol z is a nonedge parenthesis. (b) The symbol z is an edge parenthesis.

Fig. 9. The interaction between gadgets in the output of the reduction.

Lemma 4. *Let x be an edge parenthesis in either U_i or L_i and y be the mate of x. If the switch containing x (i.e., the complex U_i and L_i) is turned on (i.e., all edge parentheses are matched), the switch containing y has to be turned off (i.e., no edge parenthesis can be matched).*

Proof. Without loss of generality, we may assume that x is in U_i and it is a left parenthesis (see Fig. 9). Suppose x is aligned with z, which is a parenthesis of the same type as x. There are two cases for z.

1. The symbol z is a nonedge parenthesis (see Fig. 9(a)). Hence its mate is within L_i. However, y is located at U_j and $i < j$, and due to Lemma 3, y cannot be aligned with the mate of z. Therefore, the only way to align y is with a space. However, y is an edge parenthesis, and according to Lemma 2, the switch that contains y has to be off.
2. The symbol z is an edge parenthesis (see Fig. 9(b)). If we want to align y with a parenthesis, the only way to do this is to align y with the mate of z, due to PA3. This would indicate that there are two edges between v_i and v_j (vertices of G), one described by x and y and the other described by z and its mate. However, the graph G is simple, so there can be at most one edge between any two vertices, a contradiction. Hence, there is no way to align y with a parenthesis. Thus, y has to be aligned with a space. Accordingly, the switch containing y has to be off. □

Let I be a maximum independent set of G. Suppose there are N_i vertices of G and N_i' vertices of I that belong to Case i for $1 \leq i \leq 3$. Note that $N_1 + N_2 + N_3 = n$. From Lemmas 2–4, we have

$$\begin{aligned} S(S_1, S_2) &= 2N_1 + 2N_2 + 6N_3 + 20n(n-1) + (N_1' + N_2' + N_3') \\ &= 2n + 4N_3 + 20n(n-1) + |I| \\ &= 4N_3 + 20n^2 - 18n + |I|. \end{aligned}$$

The term $20n(n-1)$ comes from those $(n-1)$ occurrences of B complexes in S_1 and S_2, each of which contributes $20n$ matches. Hence G has an independent set of size at least K if and only if $S(S_1, S_2) \geq 4N_3 + 20n^2 - 18n + K$. On the other hand, the DPRS problem is certainly in NP. Consequently, we have Lemma 5.

Lemma 5. *The DPRS problem is NP-complete.*

Theorem 2. *The DPRD problem is NP-complete.*

Proof. The complexity of the DPRD problem is the same as the complexity of the DPRS problem. Hence this theorem follows directly from Lemma 5. □

We remark that the DPRS problem falls in a subclass of the longest arc-preserving common subsequence (LAPCS) problem (more precisely, the nested-to-nested category) proposed in [21], which has been proved NP-complete. However, it is not necessary to imply that the DPRS is trivially NP-complete since we all know that 2SAT is a special SAT but it is still polynomial-time solvable.

4. A simple 1.5-approximation algorithm

We have shown that the PRD problem is NP-hard in Section 3. In this section, we propose a 1.5-approximation algorithm for treating it.

The idea of the approximation algorithm is as follows. Consider an additional constraint for the PRD problem:

PL If a left parenthesis is aligned with a space, then its (right) mate must also be aligned with a space.

Let the parenthesis rearrangement distance according to PA1–PA3 together with PL be denoted by $D_L(P_1, P_2)$.

Similarly, consider the symmetry of PL as follows:

PR If a right parenthesis is aligned with a space, then its (left) mate must also be aligned with a space.

We use $D_R(P_1, P_2)$ to denote the parenthesis rearrangement distance according to PA1–PA3 and PR. Then we have the following lemma.

Lemma 6. *The minimum of $D_L(P_1, P_2)$ and $D_R(P_1, P_2)$ is a 1.5-approximation for $D(P_1, P_2)$ for any two nested parenthesis strings P_1 and P_2 whose lengths are equal.*

Proof. Let (A_1^*, A_2^*) be an optimal way to align P_1 and P_2 for evaluating $D(P_1, P_2)$. By definition, we have to count the number of mismatched parentheses in A_1^*. We distinguish three cases.

1. Both ends of a pair of parentheses are mismatched. Let B be the number of such pairs.
2. Only the left end of a pair of parentheses is mismatched. Let L be the number of such pairs.
3. Only the right end of a pair of parentheses is mismatched. Let R be the number of such pairs.

Then clearly we have $D(P_1, P_2) = L+R+2B$. It is not difficult to see that $D_L(P_1, P_2) \leq 2L+R+2B$ since (A_1^*, A_2^*) can be relaxed to a feasible solution for PA1–PA3 and PL by aligning L right parentheses with spaces. Similarly, $D_R(P_1, P_2) \leq L + 2R + 2B$. Therefore, we have

$$\begin{aligned} D(P_1, P_2) &\leq \min\{D_L(P_1, P_2), D_R(P_1, P_2)\} \leq L + R + 2B + \min\{L, R\} \\ &\leq L + R + 2B + (L + R)/2 \leq \frac{3}{2}(L + R + 2B) = \frac{3}{2}D(P_1, P_2). \quad \square \end{aligned}$$

In the next paragraphs, we are going to propose a simple dynamic programming algorithm that can evaluate $D_R(P_1, P_2)$ in time $O(n^4)$. The algorithm that evaluates $D_L(P_1, P_2)$ in $O(n^4)$ can also be obtained symmetrically.

Let i_1 and i_2 be indices over P_1 for $1 \leq i_1 \leq i_2 \leq 2n$ and j_1 and j_2 be indices over P_2 for $1 \leq j_1 \leq j_2 \leq 2n$. A four-dimensional array $A_R(i_1, i_2; j_1, j_2)$ is computed for evaluating $D_R(P_1, P_2)$, and finally we would get $D_R(P_1, P_2) = A_R(1, 2n; 1, 2n)$. Intuitively, $A_R(i_1, i_2; j_1, j_2)$ records the minimum cost for aligning $P_1[i_1 \ldots i_2]$ with $P_2[j_1 \ldots j_2]$ according to PA1–PA3 and rule PR such that the number of parentheses in $P_1[i_1 \ldots i_2]$ that are aligned with spaces is minimized. Notice that there could be 'free' parentheses in $P_1[i_1 \ldots i_2]$ and $P_2[j_1 \ldots j_2]$, i.e., parentheses whose mates are not within $P_1[i_1 \ldots i_2]$ and $P_2[j_1 \ldots j_2]$. In this case, all free parentheses are forced to be aligned with spaces. Accordingly, $A_R(i_1, i_2; j_1, j_2)$ can be computed by the following dynamic program, which considers all possible combinations of $P_1[i_2]$ and $P_2[j_2]$.

1. Either $P_1[i_2]$ or $P_2[j_2]$ is a left parenthesis. This left parenthesis has to be aligned with a space. Thus, let $A_R(i_1, i_2; j_1, j_2)$ be $A_R(i_1, i_2 - 1; j_1, j_2) + 1$ when $P_1[i_2]$ is a left parenthesis; otherwise, let it be $A_R(i_1, i_2; j_1, j_2 - 1)$.
2. Both $P_1[i_2]$ and $P_2[j_2]$ are right parentheses. The value for $A_R(i_1, i_2; j_1, j_2)$ is the minimum over the following four cases.
   (a) Align $P_1[i_2]$ with a space: $A_R(i_1, i_2 - 1; j_1, j_2) + 1$.
   (b) Align $P_2[j_2]$ with a space: $A_R(i_1, i_2; j_1, j_2 - 1)$.
   (c) Align $P_1[i_2]$ with $P_2[j_2]$ but not the mates: $A_R(i_1, i_2 - 1; j_1, j_2 - 1)$.
   (d) Align $P_1[i_2]$ with $P_2[j_2]$ together with the mates: $A_R(i_1, P_1[i_2'] - 1; j_1, P_2[j_2'] - 1) + A_R(P_1[i_2'] + 1, i_2 - 1; P_1[j_2'] + 1, j_2 - 1)$ where i_2' and j_2' are the positions of the mates of i_2 and j_2, respectively.

The time complexity is clearly in $O(n^4)$ since there are four indices for the A_R table and the computation for each entry takes only $O(1)$ time. To combine with Lemma 6, therefore we have the main theorem in this section.

Theorem 3. *There exists a 1.5-approximation algorithm for computing the parenthesis rearrangement distance in time $O(n^4)$.*

5. Concluding remarks

We have shown that the PRD problem is NP-hard in Section 2 and proposed a 1.5-approximation algorithm for it in Section 4. In our reduction, we reduced the maximum independent set problem restricted to simple cubic two-page graphs to this problem. It has been shown in [2] that the maximum independent set problem on planar graphs (and so for the cubic two-page graphs) admits a PTAS. Hence, our reduction provides no clue as to whether the PRD problem is APX-hard or not. We leave this question to future research.

Acknowledgements

We would like to thank the anonymous referees for their constructive comments which have greatly improved the quality of this paper.

References

[1] T. Akutsu, A relation between edit distance for ordered trees and edit distance for Euler strings, Information Processing Letters 100 (2006) 105–109.
[2] B.S. Baker, Approximation algorithms for NP-complete problems on planar graphs, Journal of the ACM 41 (1994) 153–180.
[3] J.L. Baril, J.M. Pallo, Efficient lower and upper bounds of the diagonal-flip distance between triangulations, Information Processing Letters 100 (2006) 131–136.

[4] P. Bille, A survey on tree edit distance and related problems, Theoretical Computer Science 337 (2005) 217–239.
[5] S. Cleary, Restricted rotation distance between binary trees, Information Processing Letters 84 (2002) 333–338.
[6] S. Cleary, K.S. John, Rotation distance is fixed-parameter tractable, Information Processing Letters 109 (2009) 918–992.
[7] S. Cleary, F. Luccio, L. Pagli, Refined upper bounds for right-arm rotation distances, Theoretical Computer Science 377 (2007) 277–281.
[8] S. Cleary, J. Taback, Bounding restricted rotation distance, Information Processing Letters 88 (2003) 251–256.
[9] K. Culik, D. Wood, A note on some tree similarity measures, Information Processing Letters 15 (1982) 39–42.
[10] P. Dehornoy, On the rotation distance between binary trees, Advances in Mathematics 223 (2010) 1316–1355.
[11] J. Felsenstein, Inferring Phylogenies, Sinauer Associates, 2004.
[12] M.R. Garey, D.S. Johnson, A note on "A note on 'Some simplified NP-complete graph problems'", SIGACT News 9 (1978) 17–17.
[13] M.R. Garey, D.S. Johnson, L. Stockmeyer, Some simplified NP-complete problems, in: Proceedings of the 6th Annual ACM Symposium on Theory of Computing, ACM, Seattle, WA, United States, 1974, pp. 47–63.
[14] M.R. Garey, D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, W.H. Freeman and Company, 1979.
[15] C. Germain, J. Pallo, Two shortest path metrics on well-formed parentheses strings, Information Processing Letters 60 (1996) 283–287.
[16] A. Gibbons, P. Sant, Rotation sequences and edge-colouring of binary tree pairs, Theoretical Computer Science 326 (2004) 409–418.
[17] T. Jiang, G. Lin, B. Ma, K. Zhang, A general edit distance between RNA structures, Journal of Computational Biology 9 (2002) 371–388.
[18] T. Jiang, L. Wang, K. Zhang, Alignment of trees — an alternative to tree edit, Theoretical Computer Science 143 (1995) 137–148.
[19] D.E. Knuth, The Art of Computer Programming, vol. 4, Addison-Wesley, 2006.
[20] M. Li, L. Zhang, Better approximation of diagonal-flip transformation and rotation transformation, in: The 4th Annual International Conference on Computing and Combinatorics, COCOON'98, Springer-Verlag, Taipei, Taiwan, 1998, pp. 85–94.
[21] G. Lin, Z.Z. Chen, T. Jiang, J. Wen, The longest common subsequence problem for sequences with nested arc annotations, Journal of Computer and System Sciences 65 (2002) 465–480.
[22] J.M. Lucas, D.R. Vanbaronaigien, F. Ruskey, On rotations and the generation of binary trees, Journal of Algorithms 15 (1993) 343–366.
[23] F. Luccio, L. Pagli, On the upper bound on the rotation distance of binary trees, Information Processing Letters 31 (1989) 57–60.
[24] F. Magniez, M. Rougemont, Property testing of regular tree languages, Algorithmica 49 (2007) 127–146.
[25] E. Makinen, On the rotation distance of binary trees, Information Processing Letters 26 (1988) 271–272.
[26] J. Pallo, On the rotation distance in the lattice of binary trees, Information Processing Letters 25 (1987) 369–373.
[27] J. Pallo, A distance metric on binary trees using lattice-theoretic measures, Information Processing Letters 34 (1990) 113–116.
[28] J. Pallo, An efficient upper bound of the rotation distance of binary trees, Information Processing Letters 73 (2000) 87–92.
[29] A.A. Ruiz, F. Luccio, A.M. Enriquez, L. Pagli, *k*-restricted rotation with an application to search tree rebalancing, in: Workshop on Algorithms and Data Structures, Springer, Berlin, Heidelberg, 2005, pp. 2–13.
[30] D.D. Sleator, R.E. Tarjan, W.R. Thurston, Rotation distance, triangulations, and hyperbolic geometry, Journal of American Mathematical Society 1 (1988) 647–681.
[31] K.C. Tai, The tree-to-tree correction problem, Journal of the ACM 26 (1979) 422–433.
[32] H. Touzet, A linear tree edit distance algorithm for similar ordered trees, in: The 16th Annual Symposium on Combinatorial Pattern Matching, Springer, Berlin, Heidelberg, 2005, pp. 334–345.
[33] C.S. Wu, G.S. Huang, A practical edit-distance model for RNA secondary-structure comparison, in: The 9th IEEE international conference on Bioinformatics and Bioengineering, Taichung, Taiwan, 2009.
[34] R.Y. Wu, J.M. Chang, Y.L. Wang, A linear time algorithm for binary tree sequences transformation using left-arm and right-arm rotations, Theoretical Computer Science 355 (2006) 303–314.
[35] M. Yannakakis, Embedding planar graphs in four pages, Journal of Computer and System Sciences 38 (1989) 36–67.